

Sage Lecture 2. More about Sage

We continue our exploration with Sage.

1.1 Lagrange optimization

We take a Lagrange optimization problem: Assignment 7 of lecture 23: *Find points closest to the origin on the surface $x^2 - xy + y^2 - z^2 - 1$.*

To plot the surface, in one entry field we type instructions:

```
# find points closest to origin on surface
# x^2 - x*y + y^2 - z^2 - 1
# we first plot the surface, after declaring variables:
x,y,z = var('x,y,z')
f = x^2 - x*y + y^2 - z^2 - 1
implicit_plot3d(f,(x,-2,+2),(y,-2,+2),(z,-2,+2))
```

We delete the output from the worksheet before saving to file. The file gets saved in `use_groebner.sws`.

For setting up the polynomial system in the next entry widget

```
# now we set up the system of polynomials
target = x^2 + y^2 + z^2
Lambda = var('Lambda')
e1 = diff(target,x) - diff(f,x)*Lambda
e2 = diff(target,y) - diff(f,y)*Lambda
e3 = diff(target,z) - diff(f,z)*Lambda
s = [e1, e2, e3, f]
print s
```

After the definition of the problem, we solve it:

```
# we directly try to solve the system
sols = solve([e1 == 0, e2 == 0, e3 == 0, f == 0], x,y,z,Lambda)
print sols
```

To process the output, we do some scripting in Python:

```
# some list processing to evaluate
print 'first solution :', sols[0]
y = target
for e in sols[0]: y = y.subs(e)
print y

continued:

for s in sols:
    y = target
    for e in s: y = y.subs(e)
    print 'solution', s, 'has value', y
```

Then we rename the worksheet into `lagrange_multipliers.sws` because we have solved that problem!

1.2 solving an elimination problem

In this file we describe an elimination problem, the corresponding file is `elimination.sws`.

We get through explicit formulas, but the conversion of the polynomials from the ideal to sage symbolic expressions is not so good.

```
# consider the following problem:
x,y,a,r,t,L,R = var('x,y,a,r,t,L,R')
f = (x - a)^2 + y^2 - r^2
g = (x - L*cos(t))^2 + (y - L*sin(t))^2 - R^2
print [f,g]
# let us look at a specific instance of the problem
f1 = f.subs(a == 7).subs(r = 5)
g1 = g.subs(L == 3).subs(t = pi/3).subs(R=4)
print [f1,g1]
pl1 = line([(0,0),(3*cos(pi/3),3*sin(pi/3))])
pl2 = line([(3*cos(pi/3),3*sin(pi/3)), (4+3*cos(pi/3),3*sin(pi/3))])
pl3 = line([(7,0),(7+5*cos(pi/2),5*sin(pi/2))])
pl = pl1 + pl2 + pl3
pf1 = implicit_plot(f1,(x,-10,+15),(y,-10,+10))
pg1 = implicit_plot(g1,(x,-10,+15),(y,-10,+10))
p = pl + pf1 + pg1; p.show(aspect_ratio=1)
```

Evaluate → we see two circles and three bars

```
# At the points where the two circles meet we can join the
# bars that are currently in vertical and horizontal position.
# We can solve this for particular values, we would like to solve
# this for any values of the parameters t, L, a, r, and R.
# First we turn this into an entirely algebraic problem,
# replacing sin(t) and cos(t) by s and c respectively,
# adding the equation s^2 + c^2 - 1.
c,s = var('c, s')
h = c^2 + s^2 - 1
g2 = g.subs(cos(t) == c).subs(sin(t) == s)
print [f,g2,h]
```

Evaluate → we have a polynomial system

```
# We will now go to singular, declaring a polynomial ring.
P = PolynomialRing(QQ,8,names='xyarLRcs', order='lex')
P
```

Evaluate → we see the new polynomial ring

```
# We convert our polynomial system into the polynomial ring.
# The polynomials define an ideal for which we compute a Groebner basis.
F = P(f); G = P(g2); H = P(h); print [F,G,H]
I = P.ideal(F,G,H)
gb = I.groebner_basis('libsingular:slimgb')
print 'Groebner basis :', gb
```

Evaluate → we have a triangular form for our ideal

```
# Observe the triangular form
print gb[3];
```

Evaluate → we see expression in y and other parameters, but not in x

```
# The gb[3] above depends only on y and the other parameters, but not on x.
# Once we have an expression for y, we can compute x:
print gb[2]
```

Evaluate → one expression in x

```
type(gb[2]); str(gb[2]); solve([gb[2]==0],x)
```

Evaluate → trouble with type of expressions

```
# we copy the string representation and paste it into the right hand side
# of px =
px = x*a - x*L*c - y*L*s - 1/2*a^2 + 1/2*r^2 + 1/2*L^2 - 1/2*R^2; type(px)
```

Evaluate → now we have the right type for our polynomial

```
solve([px==0],x)
```

Evaluate → we see the symbolic solution for x

```
# We copy the expression for gb[3] from the output above
# and put paste this at the right of the assignment below
# (after deleting line breaks):
py = y^2*a^2 - 2*y^2*a*L*c + y^2*L^2 - y*a^2*L*s + 2*y*a*L^2*c*s - y*r^2*L*s
- y*L^3*s + y*L*R^2*s + 1/4*a^4 - a^3*L*c - 1/2*a^2*r^2 - a^2*L^2*s^2
+ 3/2*a^2*L^2 - 1/2*a^2*R^2 + a*r^2*L*c - a*L^3*c + a*L*R^2*c + 1/4*r^4
+ r^2*L^2*s^2 - 1/2*r^2*L^2 - 1/2*r^2*R^2 + 1/4*L^4 - 1/2*L^2*R^2 + 1/4*R^4;
type(py)
```

Evaluate → the right type

```
solve([py == 0],y)
```

Evaluate → we find two solutions and see the discriminant under the sqrt.

1.3 animations

We take an example from the documentation:

```
a = animate([sin(x + float(k)) for k in srange(0,2*pi,0.3)],
             xmin=0, xmax =2*pi, figsize=[2,1])
a
```

Evaluate → Animation with 21 frames

```
a.show()
```

Evaluate → plays in continuous mode

```
a.show(iterations=1)
```

Evaluate → shows it just once